Loops

Abhiram Ranade

Outline

- while statement
- Examples
  - number of digits in a number
  - mark averaging
- do while
- for statement
- Examples
  - primality testing
  - natural log

while

while (condition) body

condition : boolean expression

body : statement

1. Evaluate condition.
2. If false, execution of statement ends.
3. If true, execute body. Then go back and execute from step 1.

Arguing correctness

- In general, a programming containing while may not terminate.
    - condition in while may never become false.

    i= 0; while(i >= 0){ i++; ….}

    - Programs with repeat always terminate
- Must argue termination and correctness.

Example: Number of digits in a number n

"Number of digits" : can we write this more formally? i.e. find x such that …

Input: non-negative integer n.

Output: Smallest $d>0$ s.t. $10^d > n$

This description of input+output is called a Specification

Algorithm from specification

Start with smallest possible d. d = 1.

Check if $10^d > n$. If so, done.

If not, try next value of d.

When we finish, will we get $10^d > n$?

Will d be smallest such value?

How do we generate 10^d?

Program

```
main_program{
 int n; cin >> n;
 int d=1, ten_power_d = 10;
 while(ten_power_d <= n){
   d++;  ten_power_d *= 10;
 }
 cout << d << endl;
}
```

Could you have written this using repeat?

Guess largest number of digits possible.

Repeat only that many times.

Details left for you to think about.

while is more powerful than repeat

while(true) body not possible using repeat.

```
repeat(n) body
```

Equivalent to:

```
int count = 0, limit = n;  // new variables
while(count < limit){ body  count++;}
```

Why worry about repeat if you have while?

If you are writing programs for special devices, e.g. a robot, you may not have full C++ but may have constraints, e.g.

- No while. Only repeat.
- Limited amount of memory. So all input data cannot be stored, but must be consumed as quickly as possible.

In general, Engineering = solve problems under constraints.

repeat is easier to learn first.

A problem impossible using repeat

Read marks of students from the keyboard and print the average. Valid marks lie between 0 and 100 (inclusive). Number of students not given explicitly. Instead, if negative value is given as mark, then it is a signal that all marks have been entered.

- 70 90 85 200 : class has 3 students.
- 75 95 99 60 88 92 77 200 : 7 students.

Algorithm idea

To calculate the average, we need the sum, and the count of how many numbers there are.

Phase 1:

Read numbers. Compute sum, count.

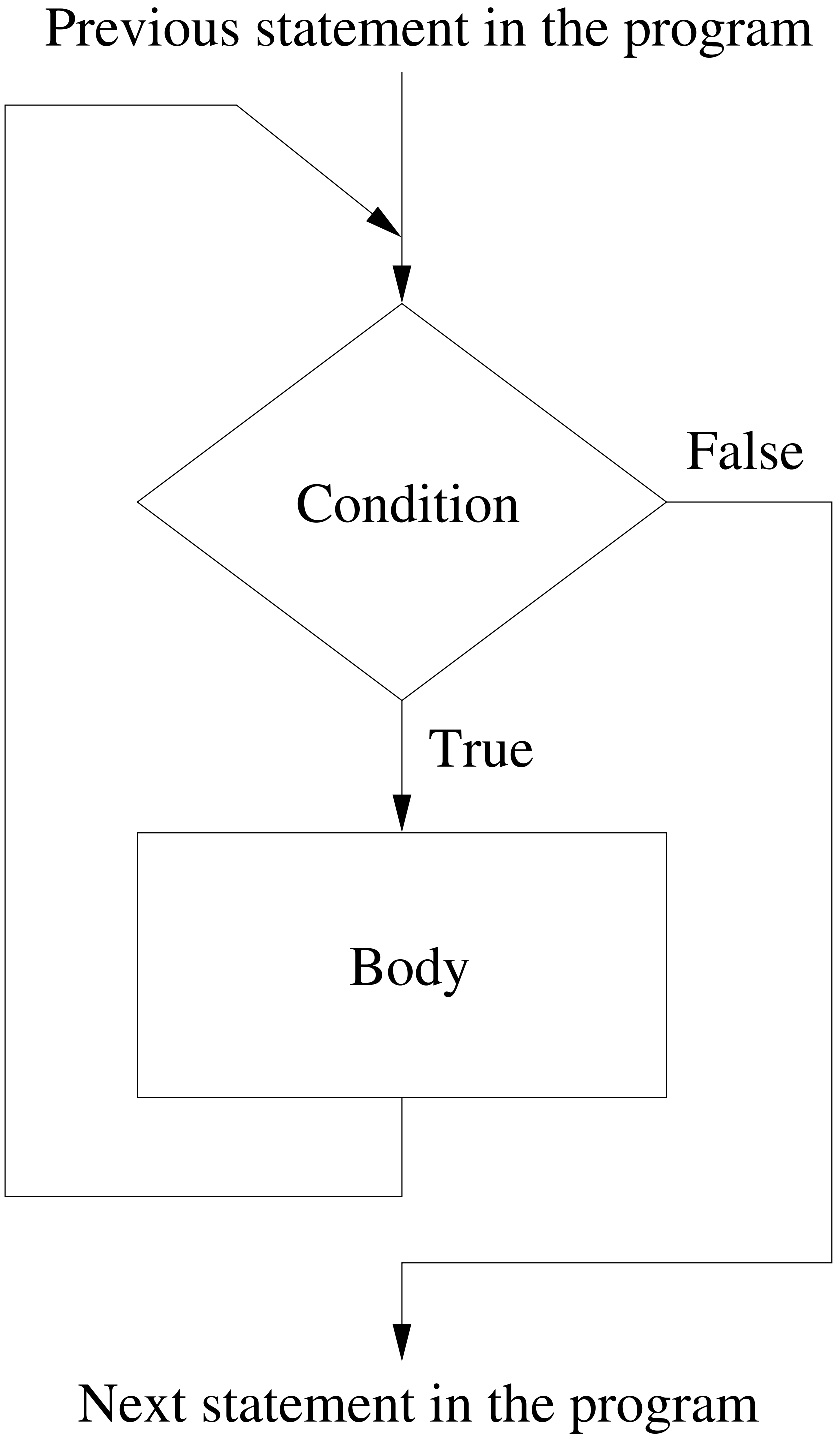
Phase 2:

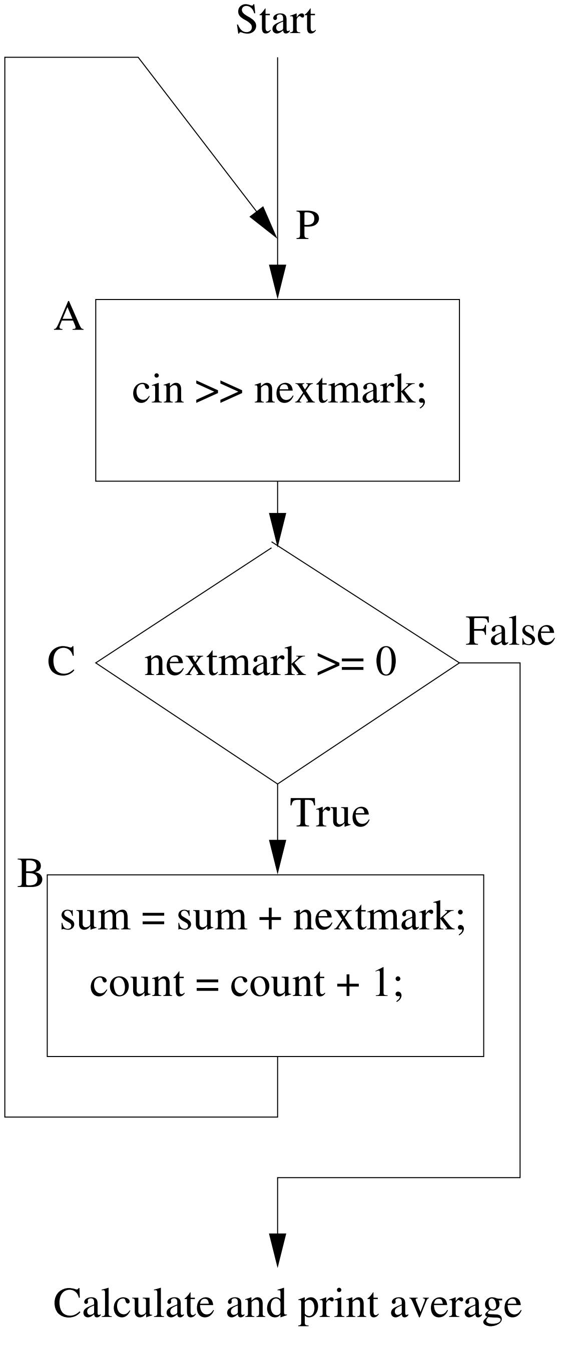
Print sum/count.

Phase 1

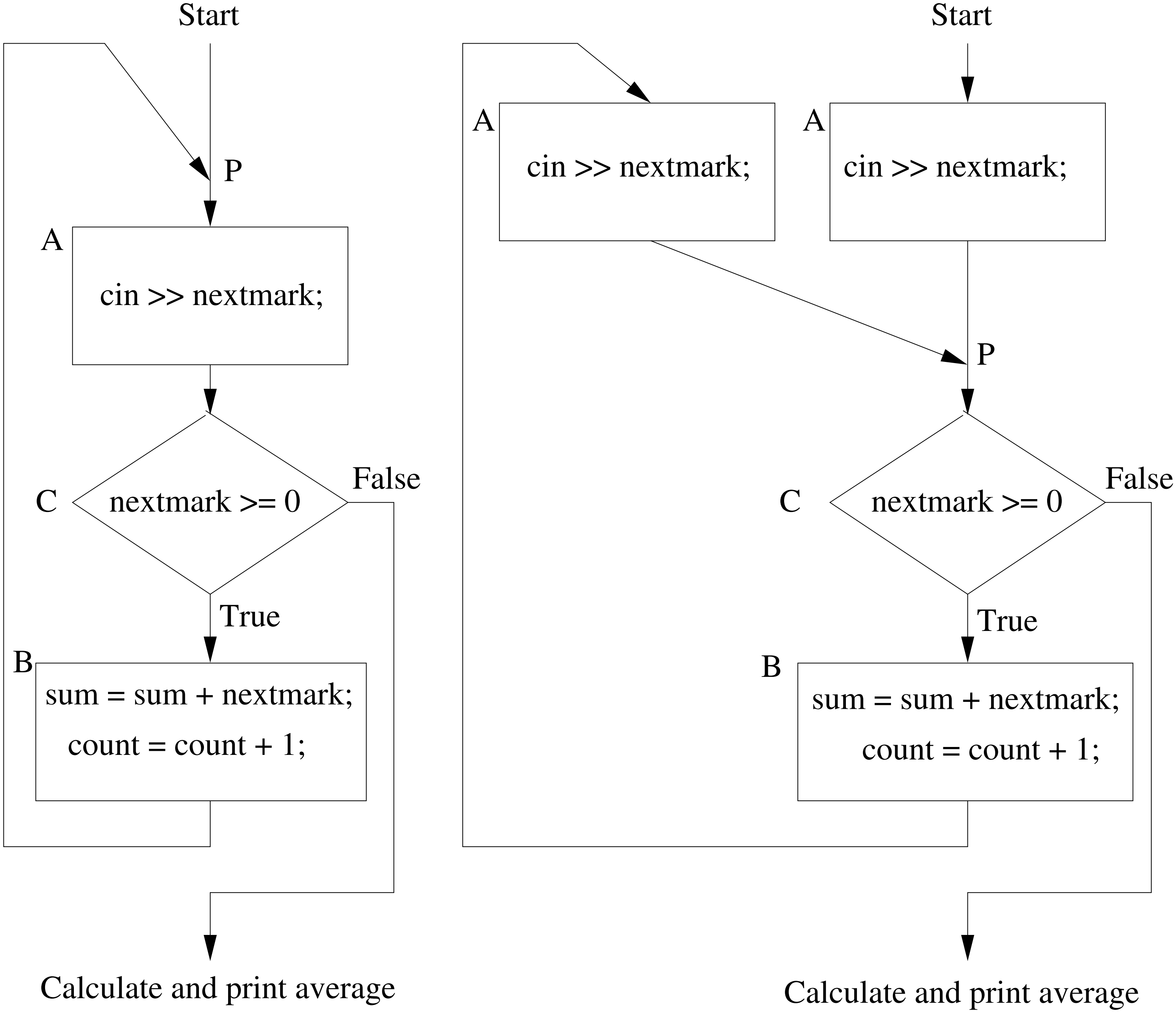
1. Read the next value into nextmark.
2. If nextmark < 0, then go to phase 2.
3. If nextmark >= 0, then add nextmark to sum. Add 1 to count.
4. Go to step 1.

How do we write this as a while?


Previous statement in the program
Condition
False
True
Body
Next statement in the program


Start
P
A
cin >> nextmark;
C
nextmark >= 0
False
True
B
sum = sum + nextmark;
count = count + 1;
Calculate and print average

(a)


Start
P
A
cin >> nextmark;
C
nextmark >= 0
False
True
B
sum = sum + nextmark;
count = count + 1;
Calculate and print average
Start
A
cin >> nextmark;
A
cin >> nextmark;
P
C
nextmark >= 0
False
True
B
sum = sum + nextmark;
count = count + 1;
Calculate and print average

(a)

(b)

Program

```
main_program{
  float nextmark, sum = 0;  int count = 0;
  cin >> nextmark;                          // A
  while(nextmark >= 0){                     // C
      sum += nextmark; count++;             // B
      cin >> nextmark;                      // A
  }
  cout << sum/count << endl;
}
```

Auxiliary Loop clauses

break

- Skips rest of current iteration.
- Goes to next statement following loop

continue

- skips rest of current iteration
- continues with next iteration.

Program 2

```
main_program{
  float nextmark, sum = 0;  int count = 0;
  while(true){
      cin >> nextmark;
     if(nextmark < 0) break;   // jump out of loop!
     sum += nextmark; count++;
  }
  cout << sum/count << endl;
}
```

Comparison of 2 programs

without break:

- cin >> .. written 2 times. In general, code duplication should be avoided.

with break:

- while condition does not express when loop terminates. Need to look inside the loop body.

Variation: ignore marks greater than 100

```
while(true){
    cin >> nextmark;
    if(nextmark > 100){
      cout << “Ignoring.\n”;
      continue;
    }
    if(nextmark < 0) break;
    sum += nextmark;  count++;
}
```

do body while condition

body : statement

condition : boolean expression

1. Execute body.
2. Evaluate condition.
3. If true, repeat from step 1.   Else, done.

The for statement

```
int i = 1;
repeat(100){
   cout <<  i  <<  ‘ ‘ <<  i*i*i  << endl;
   i++;
}
// Can be written as
for(int i=1; i<= 100; i++)
   cout << i <<‘ ‘<< i*i*i<< endl;
```

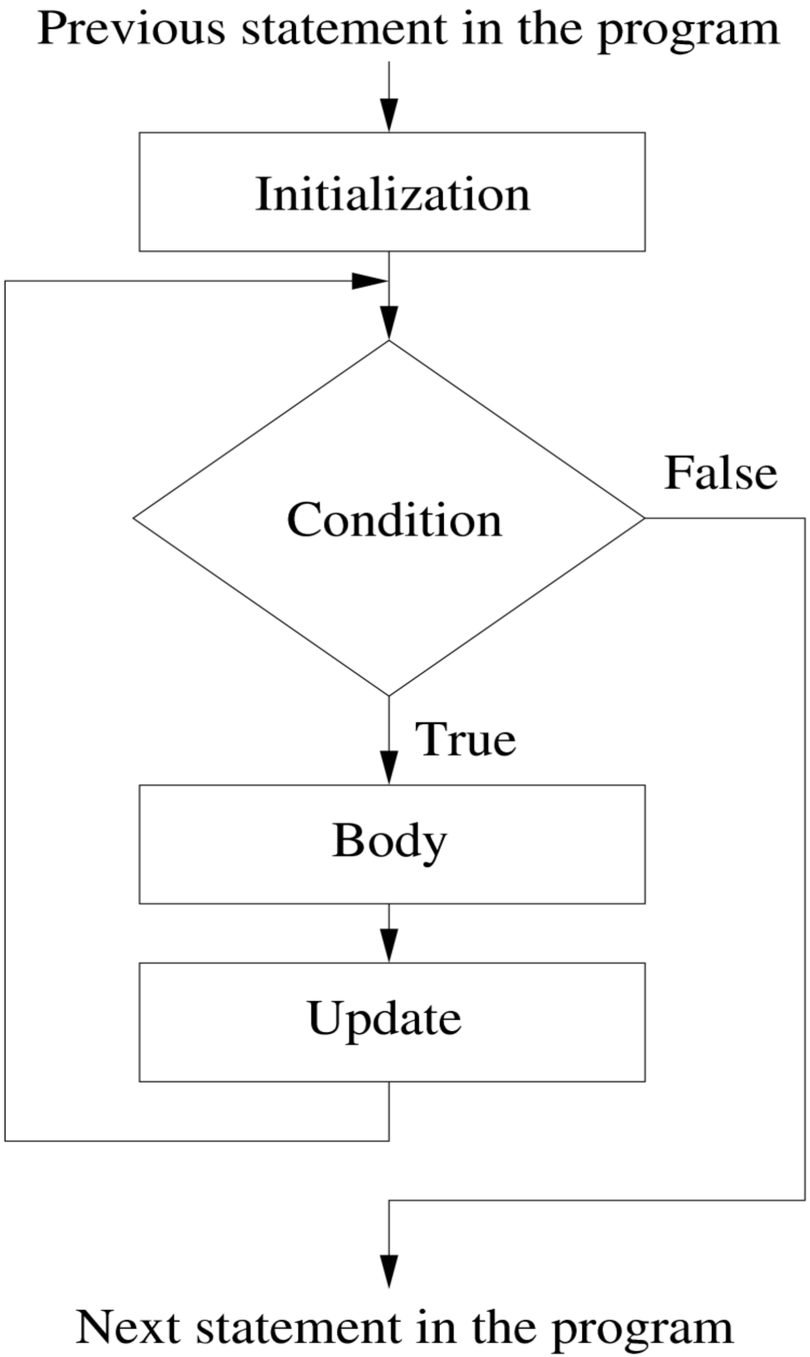
for(initialization; condition; update) body

initialization: expression. Typically assignment, or variable definition with initialization.

condition: boolean expression

update: expression. Typically assignment.

1. Execute initialization.
2. Evaluate condition.
3. If true, execute body, update, and then start again from step 2.


Previous statement in the program
Initialization
Condition
False
True
Body
Update
Next statement in the program

Primality testing

```
main_program{
  int n; cin >> n;
  bool found = false;  // factor  found?
  for(int d = 2; d < n && !found; d++)
      found = found || (n % d == 0);
  if(found) cout << “Composite.\n”;
  else cout << “Prime;\n”;
}
```

Number of digits

```
main_program{
 int n; cin >> n;
 int d, ten_power_d;
 for(d = 1, ten_power_d = 10;
      ten_power_d<= n;
      d++, ten_power_d *= 10){}
 cout << d << endl;
}
```

Remarks

- Initialization, update can be comma separated assignments.
- Body can be empty.
- If a variable is defined inside initialization, it cannot be used outside of the loop.
- Variable(s) declared in initialization: control variables.

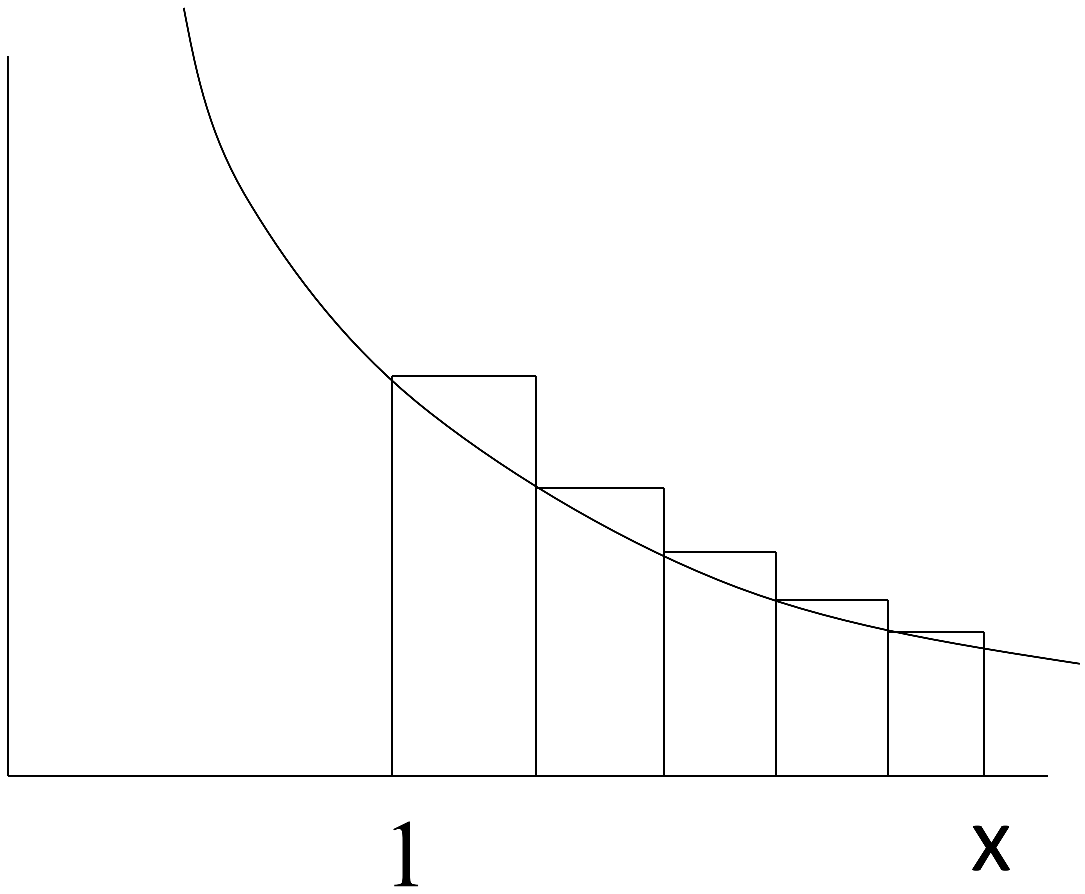
Computing ln x from the definition

Integral from 1 to x of f(x) = 1/x.

Integral = area under the curve.

Approximate area by rectangles.

Riemann Integral



How many rectangles?

More the merrier! Say $n = 1000$.

Total width of rectangles = $x - 1$.

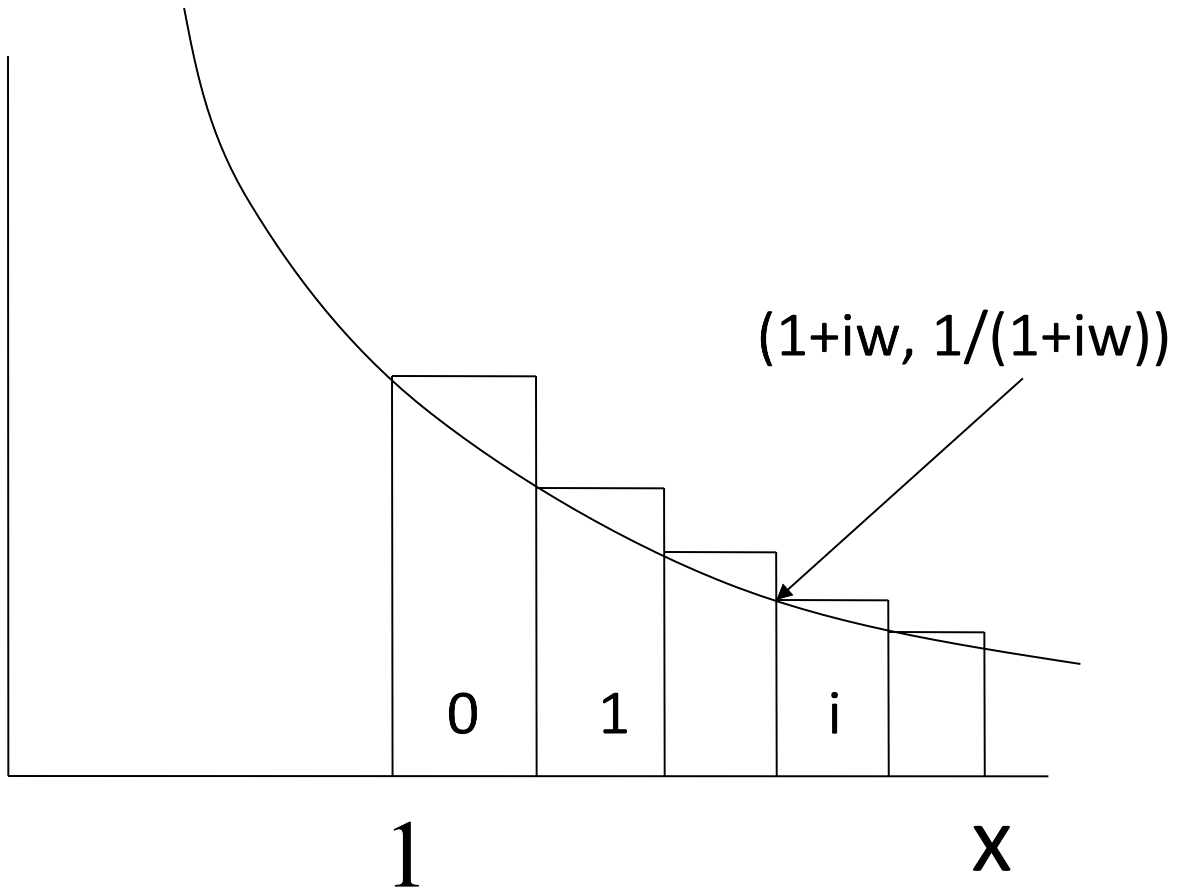
Width w of each = $(x - 1)/n$

x coordinate of left side of ith rectangle

= $1+iw$, rectangles numbered 0..999

Height of ith rectangle = $1/(1+iw)$

Riemann Integral



Program

```
main_program{
 float x; cin >> x;            // will compute ln(x)
 int n; cin >> n;              // number of rectangles
 float w = (x-1)/n;             // rectangle width
 float area = 0;               // final answer
 for(int i=0; i<n; i++)
      area += w / (1+i*w);
 cout << area << ‘ ‘ << log(x) << endl;
}  // log: built in C++ function.
```

Remarks

- Two sources of errors
  - Rectangle height = 1/x, is represented as float, correct to 7 digits only.
  - Each rectangle area approximates area under the curve.
- Large number n of rectangles:
  - More terms to add. Hence error in 1/x gets magnified.
  - Rectangles approximate area under the curve better, error reduces.
- Optimal choice of n : tradeoff.
- Tradeoff is different if numbers are double.